futuredata

FUTUREDATA Z-80

Assembler

Reference Manual

MICROCOMPUTER
SYSTEMS

Received 10/25/78

FUTUREDATA Z-80

Assembler

Reference Manual

FUTUREDATA COMPUTER CORP.
11205 S. La Cienega Blvd.
Los Angeles, CA 90045

Copyright 1977, FUTUREDATA COMPUTER CORP.

TABLE OF CONTENTS

		Page
1.0	INTRODUCTION	1-1
2.0	ASSEMBLER STATEMENTS	2-1
	2.1 LABEL FIELD	2-1
	2.1.1 Assembler Operand Symbols	2-2
	2.2 OPCODE FIELD	2-2
	2.3 OPERAND FIELD	2-3
	2.3.1 Register Operands	2-3
	2.3.2 Restart Address Data	2-4
	2.3.3 Immediate Data.	2-4
	2.3.4 Address Data	2-4
	2.3.5 Address Data (Index-offset)	2-4
	2.3.6 Address Data (Relative).	2-4
	2.3.7 Hexadecimal Constants.	2-4
	2.3.8 Decimal Constants	2-5
	2.3.9 Current Location Counter.	2-5
	2.3.10 ASCII Characters	2-5
	2.3.11 Symbol Values	2-5
	2.3.12 Register Symbols	2-6
	2.3.13 Expressions	2-6
	2.4 Z-80 INSTRUCTION TYPES	2-7
	2.4.1 One Byte, No Operands	2-7
	2.4.2 One-Byte, One Register Operand . . .	2-7
	2.4.3 One-Byte, Two Register Operands . .	2-7
	2.4.4 One-Byte, Condition Operand	2-8
	2.4.5 One-Byte, Restart Address Operand .	2-8
	2.4.6 Two-Byte, No Operands	2-9
	2.4.7 Two-Byte, One Register Operand . . .	2-9
	2.4.8 Two-Byte, Two Register Operands . .	2-10
	2.4.9 Two-Byte, Register and Immediate Operands	2-10
	2.4.10 Two-Byte, Relative Address Operand.	2-10
	2.4.11 Two-Byte, Bit and Register Operands	2-11
	2.4.12 Two-Byte, Condition and Relative Address Operands	2-11
	2.4.13 Two-Byte, Register and I/O Address Operands	2-11
	2.4.14 Three-Byte, Index+Offset Operand . .	2-12
	2.4.15 Three-Byte, Register and Index+Offset Operands	2-12

2.4.16 Three-Byte, Register and Immediate Address Operands 2-12
2.4.17 Three-Byte, Register and Address Operands 2-13
2.4.18 Three-Byte, Address Operand 2-13
2.4.19 Three-Byte, Condition and Address Operands 2-13
2.4.20 Four-Byte, Register and Immediate Address Operands 2-14
2.4.21 Four-Byte, Index+Offset and Immediate Operands 2-14
2.4.22 Four-Byte, Index+Offset Operand . . 2-14
2.4.23 Four-Byte, Bit and Index+Offset Operands 2-15
2.4.24 Four-Byte, Register and Address Operands 2-15

2.5 ASSEMBLER PSEUDO-INSTRUCTIONS 2-15

2.5.1 END 2-15
2.5.2 SPC 2-16
2.5.3 EJE 2-16
2.5.4 EQU 2-16
2.5.5 ORG 2-16
2.5.6 DC 2-17
2.5.7 DS 2-18
2.5.8 PRNT ON 2-18
2.5.9 PRNT OFF 2-18

2.6 COMMENTS 2-18

3.0 ERROR FLAGS 3-1

3.1 N 3-1
3.2 O 3-1
3.3 P 3-1
3.4 U 3-1
3.5 R 3-2
3.6 D 3-2
3.7 A 3-2

4.0 ASSEMBLER EXECUTION 4-1

4.1 LOADING THE ASSEMBLER 4-1

4.2 RUNNING PASS 1 4-1

4.3 RUNNING PASS 2 4-2

4.4 OPERATING SUGGESTIONS 4-4

1.0 INTRODUCTION

The FUTUREDATA Z-80 Assembler runs on Microsystem 10, 15, 20 and 30 microcomputer development systems and generates binary-load files from a source program written in symbolic assembly language.

Programs written in assembly language make use of mnemonic names for the machine instructions and special assembler pseudo-instructions. The assembler translates these mnemonic instructions into the binary machine codes. Symbolic labels are used for addresses in the source program; these symbolic addresses are also converted into absolute binary machine addresses by the assembler.

The assembler processes the source program in two passes through the input. During the first pass, the assembler builds a symbol table which equates absolute binary values with the symbolic names used in the source program. During pass 2, the assembler uses the symbol table and translates each source program statement into loadable binary object data.

A number of options are available during pass 2. A listing of the program can be requested for display on the CRT. Any errors detected by the assembler are flagged in this listing. A hardcopy of the program listing can be requested, and a loadable object tape or disk file can optionally be written during pass 2 also.

The assembler will run in a 16K memory configuration. However, if more memory is available, the assembler will automatically configure itself for the larger system. The 16K assembler has a symbol table capable of holding approximately 1000 symbols averaging 4 characters in length. The symbol table will hold more shorter symbols and few longer symbols.

2.0 ASSEMBLER STATEMENTS

Assembly programs are composed of three types of statements: (1) executable statements, (2) pseudo statements or instructions, and (3) comment statements. Each statement is one line of assembly code.

Assembly language statements are made up of four fields: (1) label field, (2) operation code field, (3) operand field, and (4) optional comment field.

The label field optionally contains a symbolic name which may be used when referencing the statement from other statements in a program.

The operation code or opcode field contains a Z-80 instruction mnemonic name or an assembler pseudo-instruction name.

The operand field contains additional data (e.g., immediate data or address) required for the specified opcode. Some instructions do not require any operand data.

The comment field is the last field and allows for annotation of the program. This field is not processed by the assembler, but is printed in the program listing.

The assembler statements have free fields, which means that the fields of the statement do not have rigidly defined lengths or positions. Instead, each field is separated from the next by one or more blanks. However, it must be noted that if no label is provided on a statement, at least one blank must still precede the opcode field.

The fields of the assembler statement will be represented diagrammatically in later discussions as follows:

Label Opcode Operand Comments

The maximum length of an assembler statement is 79 characters. Paragraph 2.6 describes the method for extending assembler statements beyond 40 characters.

2.1 LABEL FIELD

A symbolic label may be supplied on statements which must be referenced from other statements of a program. The symbolic label is optional and need not be coded in every statement. When the label field is omitted, as is usually the case, at least one blank

must precede the opcode field.

Valid symbolic labels are 1 to 8 characters long. The first character of the label must always be alphabetic, but other characters may be alphabetic or numeric. The delimiter between the label field and the opcode field can be either a blank " " or a colon ":". Examples of valid labels are:

```
LOC1
LOOP
MXLENGTH
A123
LOC1:
MXLENGTH:
```

During pass 1, the assembler assigns values to symbolic labels. The symbols and their assigned values are stored in a symbol table in the microcomputer memory. Since the symbol table is designed so that shorter symbols use less space and longer symbols use more space, it is advised that short symbol names be used where possible. This will conserve space in the symbol table and will allow a greater number of symbols to be accommodated.

Normally the value assigned to a symbol is the absolute memory location of the instruction statement on which the label occurs. However, a special assembler pseudo-instruction, EQU, described in detail in Section 2.5.4, allows for the symbol appearing on the statement to be assigned the value of the operand field.

2.1.1 Assembler Operand Symbols

It is recommended that the user avoid symbols that may conflict with valid Z-80 operands. These include register designators such as: A, B, BC, SP, or AF; and condition codes such as Z, NZ, P, or M.

2.2 OPCODE FIELD

The opcode field contains the name of a Z-80 machine instruction to be generated by the statement, or an assembler pseudo-instruction. Each of the machine instructions of the Z-80 has been assigned a 3 to 4-character mnemonic name. The instruction set mnemonics used by the FUTUREDATA Z-80 Assembler are the same mnemonics assigned by Zilog. One of the main functions of the assembler is to convert the easy-to-remember symbolic instruction names into the binary machine codes.

Also appearing in the opcode field are assembler pseudo-instructions. These are mnemonic names which do not generate executable instructions, but instead command the

assembler to perform special functions. The assembler pseudo-instructions will be detailed in Section 2.5.

2.3 OPERAND FIELD

The operand field is used to supply parameters (e.g., immediate data or addresses) which are required for the assembly of some instructions. The operand field is separated from the opcode by one or more blanks. Depending upon the opcode, the operand field may be omitted or may contain one or more data elements. The data elements within the operand field are separated by commas only. Since a blank is used to separate the operand field from the comment field, a blank inadvertently included in the operand field will cause the data following the blank to be ignored as a comment.

There are six types of data elements which may be required in the operand field, depending upon the opcode. These data types are:

Data Type	Implied Length
Register	3 bits
Restart Address	3 bits
Immediate data	8 bits (1 byte)
Address	16 bits (2 byte)
Offset	8 bits (1 byte)
Relative address	8 bits (1 byte)

There are six ways of specifying data as follows:

Decimal constant
Hexadecimal constant
Current location counter
ASCII character
Value of symbolic names
Expressions

All operands are computed as 16-bit values. Only the number of bits implied by the data type are used in assembling the instruction. The operand value is taken from the low-order part of the computed value and high-order data is ignored.

2.3.1 Register Operands

Register operands are required on most arithmetic instructions and some load and store instructions. A register operand specifies the machine register to be incorporated into the assembled instruction. For convenience in coding register operands, the symbols listed in Paragraph 2.3.12 are permanently defined and will modify the machine code as required.

2.3.2 Restart Address Operands

There are 8 restart addresses (0 through 7). These select restart addresses 0 through 56 (decimal).

2.3.3 Immediate Data

Immediate operands are required on all immediate instructions. Immediate operands are either 8 bits in length and have values between 0 and 255 or 16 bits in length and have values between 0 and 65,535. The class of instruction determines the length of the immediate operand. Immediate operands may be specified by any of the acceptable ways of defining data values.

2.3.4 Address Data

Address operands are required on jump, call, direct address load/store, and double-register load instructions. Address operands are 16 bits long and have values between 0 and 65,535.

2.3.5 Address Data (Index-offset)

Offset address operands are required on all instructions using the index-registers as base addresses. These include indexed load, store, and arithmetic instructions. Index operands are eight bits long and have values between -128 and +127.

2.3.6 Address Data (Relative)

Relative address operands are required on all conditional branch instructions, and unconditional branch instructions. The relative operands are eight bits in length and have values between -125 and +129.

2.3.7 Hexadecimal Constants

Hexadecimal constants are made up of the hexadecimal digits 0-9 and A-F and must be prefixed by an "X" and enclosed in single quotes have an "H" as a suffix. When the suffix "H" form is used the first element of the constant must be a number between 0 and 9.

Single-byte values may be one or two digits long while two-byte (address) values may be one to four digits in length. When less than the maximum number of digits are used, the numbers are treated as if leading zeros were supplied; i.e., they are right justified.

Examples of single-byte hexadecimal constants are:

```
X'1' (= X '01')
X'A3'
0AFH
```

Examples of two-byte (address) constants are:

```
X'103' (= X '0103')
X'BF2A'
0BADEH
```

2.3.8 Decimal Constants

Decimal constants are simply numbers made up of the digits 0-9. Single-byte decimal values may range from 0 to 255. Two-byte decimal values may range from 0 to 65,535.

2.3.9 Current Location Counter

The symbols "*" or "$" are used to represent the current value of the location counter. This is the address of the first byte of the assembled statement in which the "*" or "$" occurs.

2.3.10 ASCII Characters

Character constants store the code for symbolic characters as defined by the American Standard Code for Information Interchange (ASCII). An ASCII character constant must be enclosed in single quotes. A single-quote character is represented by two single quotes occurring sequentially.

An ASCII character occupies a single byte. The ASCII code is a 7-bit code; the eighth (high-order) bit is zero when stored in an 8-bit byte.

Examples of ASCII character constants are:

```
'A'
'*'
''''
```

2.3.11 Symbol Values

Values assigned to symbolic names may be used as operands in addition to the constants described in previous sections. During pass 1 of the assembly, each symbol occurring in the label field is assigned a numeric value. The value assigned to a symbol is either the address of the statement at which it occurs or, for the EQU instructions (see Section 2.5), the value of the operand is assigned to the symbol.

Values of symbols are stored in the symbol table during pass 1. During pass 2, whenever the symbol is used in an operand, the assigned value is retrieved from the symbol table.

2.3.12 Register Symbols

As noted in Section 2.1.1, several symbols are permanently defined by the assembler. These symbols are used when specifying a register operand. The valid Z-80 register designators are:

A	BC
B	DE
C	HL
D	SP
E	IX
H	IY
L	
AF	

2.3.13 Expressions

Operands may also be defined as expressions composed of the constants or symbols described in Sections 2.3.1 to 2.3.9.

Two binary operators, plus and minus, allow for the combinations of values. Multiple operators are allowed, and such expressions are evaluated from left to right. Parentheses may not be used for grouping elements for evaluation. Examples of expressions are:

```
ALPHA+25
*-3
X'36'-1
ALPHA-X'2E'+1
```

The value of the expression

```
2+6-3
```

is 5, as a result of the left-to-right evaluation.

Expressions are always evaluated as 16-bit quantities, though fewer bits may be required for some operands.

Several unary operators are also provided. They are:

```
H(expression)
L(expression)
```

The H (high-byte) and L (low-byte) operators are used primarily for single-byte operands. The "H" operator extracts the

high-order byte of the constant, symbol, or expression in parenthesis. Similarly, the "L" operator extracts the low-order byte of the constant, symbol, or expression in parenthesis. For example:

```
H(ALPHA)
L(X'2E35'+2)
```

2.4 Z-80 INSTRUCTION TYPES

The Z-80 Assembler has one-byte, two-byte, three-byte and four byte instructions. These may be classified into twenty four distinct types as follows.

2.4.1 One Byte, No Operands

Some one-byte instructions do not require any operands.

For example

```
EXX
```

Instructions of this type are:

CCF	EXX	RRA
CPL	HALT	RRCA
DAA	NOP	RET
DI	RLCA	SCF
EI		

2.4.2 One-Byte, One-Register Operand

Many one-byte instructions require a register operand. This operand is specified as one of the predefined register symbols (A, B, C, D, E, H, L, BC, DE, HL, SP, (HL)).

Example:

```
DEC B
```

Instructions of this type are:

DEC r	INC r	JP (HL)
DEC (HL)	INC (HL)	PUSH qq
DEC ss	INC ss	POP qq

Where ss can be BC, DE, HL, or SP; and qq can be AF, BC, DE, or HL.

2.4.3 One-Byte, Two-Register Operands

Many one byte instructions require two register operands. These include exchange, arithmetic, and load instructions.

Example:

```
ADD A,C
```

Instructions of this type are:

```
ADD A,r         EX DE,HL        LD (DE),A
ADD A,(HL)      EX AF,AF'       OR A,r
ADD HL,ss       EX (SP),HL      OR A,(HL)
ADC A,r         LD r,s          SBC A,r
ADC A,(HL)      LD r,(HL)       SBC A,(HL)
AND A,r         LD (HL),r       SUB A,r
AND A,(HL)      LD A,(BC)       SUB A,(HL)
CP A,r          LD A,(DE)       XOR A,r
CP A,(HL)       LD (BC),A       XOR A,(HL)
                LD SP,HL
```

Where ss is BC, DE, HL or SP.

2.4.4 One-Byte, Condition Operand

Conditional return instructions require a condition code as an operand.

Example:

```
RET             NZ
```

Instructions of this type are:

```
RET C           RET M           RET Z
RET NC          RET PE          RET NZ
RET P           RET PO
```

2.4.5 One-Byte, Restart Address Operand

The Restart instruction requires an operand that evaluates to a number between 0 and 7.

Example:

```
RST             5
```

The only instruction of this type is:

```
RST n
```

2.4.6 Two-Byte, No Operands

Some of the I/O and return from interrupt instructions fall in this class. All of the block instructions and interrupt mode-set instructions are also in this class.

Example:

```
                    INDR
```

Instructions of this type are:

IND	INDR	INI
INIR	OUTD	OTDR
OUTI	OTIR	
CPI	CPDR	LDD
CPIR	LDI	LDDR
CPD	LDIR	
IM0	IM1	IM2
RETI	RETN	
RLD	RRD	NEG

2.4.7 Two-Byte, One Register Operand

Index register pushes, pops and index register vectored jumps, index register increments and decrements are two-byte and one operand. One form of the rotate and shift instructions is also of this type.

Example:

```
                    PUSH            IY
```

Instructions of this type are:

JP (IX)	POP IX	PUSH IX
JP (IY)	POP IY	PUSH IY
RLC r	RRC r	SLA r
RLC (HL)	RRC (HL)	SLA (HL)
RL r	RR r	SRA r
RL (HL)	RR (HL)	SRA (HL)
SRL r	SRL (HL)	
INC IX	DEC IX	

```
INC IY          DEC IY
```

2.4.8 Two-Byte, Two Register Operands

Instructions to transfer data from the Z-80 index, interrupt vector, and refresh registers require two bytes and two operands. Some of the double precision add and subtract instuctions are in this category, as well as some of the I/O instructions.

Example:

```
                LD          SP,IX
```

Instructions of this type are:

```
LD A,I          LD SP,IX        ADC HL,ss
LD I,A          LD SP,IY        SBC HL,ss
LD A,R          EX (SP),IX      ADD IX,pp
LD R,A          EX (SP),IY      ADD IY,rr
IN r,(C)        OUT r,(C)
```

Where ss can be BC, DE, HL or SP; pp can be BC, DE, IX or SP; and rr can be BC, DE, IY or SP.

2.4.9 Two-Byte, Register and Immediate Operands

All two-byte immediate instructions require a one-byte immediate operand. Though the operand, which may be a constant, symbol, or expression, will be evaluated as 16-bit value, only the low-order eight bits will be used for the immediate data value.

Example:

```
                ADC         A,26
```

Instructions of this type are:

```
ADC A,n         OR A,n          LD (HL),n
ADD A,n         SBC A,n         LD r,n
AND A,n         SUB A,n
CP A,n          XOR A,n
```

2.4.10 Two-Byte, Relative Address Operand

Two relative jump instructions are of this form. The relative address may be a symbol, constant or expression. The relative address operand will be combined with present contents of the location counter plus 2 to form the effective address. The relative address operand must evaluate to a number between -125 and +129.

Example:

```
                    DJNZ            LOOP
```

Instructions of this type are:

```
        DJNZ e          JR e
```

2.4.11 Two-Byte, Bit and Register Operands

The Bit test, Bit set and Bit reset instructions allow this form.

Example:

```
                    BIT             3,(HL)
```

Instructions of this type are:

```
        BIT b,r         SET b,r         RES b,r
        BIT b,(HL)      SET b,(HL)      RES b,(HL)
```

Where b is the bit number (0-7) and r is the register designator (B, C, D, E, H, OR L).

2.4.12 Two-Byte Condition and Relative Address Operands

Relative jump instructions require a condition code and relative address. The relative address may be a symbol, constant or expression. The relative address operand will be combined with the present contents of the location counter plus 2 to form the effective address. The relative address operand must evaluate to number between -125 and +129.

Example

```
                    JR              NZ,LOOP3
```

Instructions of this type are:

```
        JR C,e          JR Z,e
        JR NC,e         JR NZ,e
```

2.4.13 Two-Byte, Register and I/O Address Operands

The 8080 compatible form of of the I/O instruction is two bytes with register and I/O address operands. The I/O address operand may be a symbol, constant or expression. The operand will be evaluated as a 16-bit value though only the low order 8 bits will be used.

Example:

```
        TTYOUT          OUT             PORT1
```

Instructions of this type are:

```
        IN A,(n)        OUT (n),A
```

2.4.14 Three-Byte, Index+Offset Operand

Increment and Decrement instructions can be used to affect memory locations whose addresses are in the IX or IY registers.

Example:

```
        INCRM           INC             (IX+COUNT5)
```

Instructions of this type are:

```
        INC (IX+d)      DEC (IX+d)
        INC (IY+d)      DEC (IY+d)
```

2.4.15 Three-Byte, Register and Index+Offset Operands

Load instructions and arithmetic instructions using the IX and IY registers plus offsets to compute the effective address are provided. In this case the offset may be a symbol, constant or expression. The offset will be evaluated as a signed 8 bit value (-128 to +127).

Example:

```
                        ADD             A,(IX+X'24')
```

Instructions of this type are:

```
        LD (IX+d),r     LD (IY+d),r
        LD r,(IX+d)     LD r,(IY+d)

        ADD A,(IX+d)    CP A,(IX+d)     SUB A,(IX+d)
        ADD A,(IY+d)    CP A,(IY+d)     SUB A,(IY+d)
        ADC A,(IX+d)    OR A,(IX+d)     XOR A,(IX+d)
        ADC A,(IY+d)    OR A,(IY+d)     XOR A,(IY+d)
        AND A,(IX+d)    SBC A,(IX+d)
        AND A,(IY+d)    SBC A,(IY+d)
```

2.4.16 Three-Byte, Register and Immediate Address Operands

Three-byte immediate address instructions require a two-byte

(16-bit) address operand. The operand may be a symbol, constant or expression.

Example:

```
                    LD              HL,LIMIT
```

Instructions of this type are:

```
                    LD dd,nn
```

Where dd is BC, DE, HL or SP.

2.4.17 Three-Byte, Register and Address Operands

Three-byte register and address instructions require a register operand and a two-byte extended address. The address operand may be a symbol, constant or expression.

Example:

```
        GETCH       LD              HL,(TEMP)
```

Instructions of this type are:

```
        LD A,(nn)       LD HL,(nn)
        LD (nn),A       LD (nn),HL
```

2.4.18 Three-Byte, Address Operand

Three-byte instructions require a two-byte (16-bit) address operand. The operand may be a symbol, constant, or expression.

Example:

```
                    JP              ALPHA
```

Instructions of this type are:

```
        JP nn           CALL nn
```

2.4.19 Three-Byte, Condition and Address Operands

Conditional Jumps and Calls require condition code and address operands. The address may be an expression, constant or symbol.

Example:

```
                    JP              NZ,ADDR3
```

The instructions of this type are:

JP C,nn	JP P,nn	CALL PE,nn
JP NC,nn	JP M,nn	CALL PO,nn
JP Z,nn	CALL C,nn	CALL P,nn
JP NZ,nn	CALL NC,nn	CALL M,nn
JP PE,nn	CALL Z,nn	
JP PO,nn	CALL NZ,nn	

2.4.20 Four-Byte, Register and Immediate Address Operands

The index register load immediate instructions require register designation and immediate 16 bit address operands. The immediate operand may be a symbol, constant or expression. It will evaluate to an unsigned 16 bit number.

Example:

```
INIT            LD              IX,TBL
```

Instructions of this type are:

LD IX,nn	LD IY,nn

2.4.21 Four-Byte, Index+Offset and Immediate Operands

The load immediate to memory (using the index registers as address pointers) instructions are of this form. Both the offset and the immediate operands may be symbols, constants or expressions.

Example:

```
                LD              (IX+25),'B'
```

Instructions of this type are:

LD (IX+d),n	LD (IY+d),n

2.4.22 Four-Byte, Index+Offset Operand

The rotate and shift instructions can operate on memory locations whose addresses are contained in the index registers. The offset may be a symbol, contstant or expression.

Example:

```
                RLC             (IX+X'33')
```

Instructions of this type are:

```
RLC (IX+d)      RRC (IY+d)      SRA (IX+d)
RLC (IY+d)      RR (IX+d)       SRA (IY+d)
RL (IX+d)       RR (IY+d)       SRL (IX+d)
RL (IY+d)       SLA (IX+d)      SRL (IY+d)
RRC (IX+d)      SLA (IY+d)
```

2.4.23 Four-Byte, Bit and Index+Offset Operands

The bit manipulation instructions can use the index registers as address operands. The offset operand may be a symbol, constant or expression. The bit operand may also be a symbol, constant or expression. It must evaluate to a number between 0 and 7.

Example:

```
FLAGON          SET             FLAG,(IY+54)
```

Instructions of this type are:

```
BIT b,(IX+d)    SET b,(IX+d)    RES b,(IX+d)
BIT b,(IY+d)    SET b,(IY+d)    RES b,(IY+d)
```

2.4.24 Four-Byte, Register and Address Operands

Direct address load and store instructions are of this type. The address operand may be a constant, symbol or expression.

Example:

```
                LD              BC,(TABLE3)
```

Instructions of this type are:

```
LD IX,(nn)      LD (nn),IX      LD dd,(nn)
LD IY,(nn)      LD (nn),IY      LD (nn),dd
```

Where dd is BC, DE or SP.

2.5 ASSEMBLER PSEUDO-INSTRUCTIONS

There are eight instruction mnemonics which are not Z-80 machine instructions, but which are treated as commands to the assembler. These assembler commands perform extremely useful functions.

2.5.1 END

The END statement is used to signal the end of assembler input statements. This statement must always be supplied as the last statement of a program.

The operand field of the END statement should contain an address operand. This address specifies the program entry point. That is, when the program is loaded by the monitor, the entry pointer will be set to this address. The execute command can then be used to begin execution at the address specified as the entry point.

2.5.2 SPC

The space instruction is used to increase the readability of the assembler listing. This statement is replaced by a blank line in the listing. No operands are processed.

2.5.3 EJE

The eject instruction is also used to format the program listing. Eject causes the assembler to skip to the top of the next page. No operands are processed.

2.5.4 EQU

The equate instruction is used to assign a value to a symbolic name. The symbol in the label field is assigned the value of the operand field, evaluated during pass 1. Because the value is assigned to the symbol during pass 1, it may be used like any other symbol and may be referenced in the operand fields of statements occurring before and after the equate in which it is defined. However, there is a restriction on the symbols that may be used in the operand field. Symbols referenced by the EQU instruction must have been defined (i.e., occur in the label field) in some statement prior to the equate statement.

Examples:

```
DATA    EQU    X'3F'
DATA1   EQU    DATA+26
```

2.5.5 ORG

The origin pseudo-instruction provides the capability for specifying the address in memory at which the subsequent statements are assembled. Normally sequential statements are placed in sequential memory locations during the assembly process. The origin instruction alters this by specifying that the subsequent instruction should be assembled at the address specified in the operand field.

The operand for the ORG instruction may be a constant, symbol, or expression representing a 16-bit machine address. As with EQU, symbols used in the operand field must be defined in a previous statement.

If no ORG instruction is given as the first statement of a program, the assembler begins with location 0.

Examples:

```
        ORG     X'1000'
        ORG     *+10
```

2.5.6 DC

The DC pseudo-instruction is used to define constant data to be placed in memory. Both single-byte and two-byte (address) elements may be generated by the DC instruction.

The operand field of the DC instruction may contain one or more valid operands; i.e., constants, symbols, or expressions. When a list of operands is specified, each operand must be separated from the next by a comma. A blank delimits the end of the list, consequently no blanks are allowed within or between operands, except inside an ASCII string constant. The operands within a list are stored in sequential memory locations.

While operands for other statements only allow a single-byte ASCII character constant, the DC instruction allows ASCII character strings of varying length. As with the single-character constant described in Section 2.3.7, ASCII strings are enclosed in single quotes. A single quote within the string is represented by two single quotes occurring sequentially. Each ASCII character occupies a single byte of memory. The characters within the string are stored in sequential bytes of memory.

Operands of the DC instruction are normally each stored in a single byte of memory. A two-byte operand can be specified by the "A" operator or the "B" operator. An operand preceded by an "A" and enclosed in parenthesis will be stored as a two-byte address. An operand preceded by a "B" and enclosed in parenthesis will also be stored as a two-byte address, but the low-order byte will be stored ahead of the high-order byte. Two-byte operands can be intermixed with single-byte operands in the operand list of the same DC statement.

Examples of DC instructions are as folows:

```
        DC      X'23'
        DC      ALPHA+1,21,'ABC'
LOC     DC      H(BETA),A(ALPHA+X'23')
        DC      'DON''T'
        DC      B(BETA)
        DC      A(X'3F21'),X'21'
```

2.5.7 DS

The DS pseudo-instruction is used to reserve memory space, but without initializing it to a specific value as with the DC pseudo-instruction.

The operand field of the DS statement contains the number of bytes of memory to reserve. This operand may be a constant symbol, or expression, but like the EQU and ORG statements, any symbols used in the operand must have been defined in a previous statement.

Example:

```
DATA        DS          25
```

2.5.8 PRNT ON

The PRNT ON and PRNT OFF pseudo instructions allow for selective listing of the assembler output. PRNT ON turns on the hard-copy listing device.

2.5.9 PRNT OFF

The PRNT OFF pseudo instruction turns off the hard copy listing device.

2.6 COMMENTS

If the first character in a statement is an "*" or a ";" then that statement will be treated as a comment. Comments, of course, produce no effect on the assembly of the program. Examples of comment statements are as follows:

```
        *
        *THIS PROGRAM COMPUTES THE SQUARE
        *ROOT OF A+B-2
        *
```

or

```
        ;
        ;THIS PROGRAM COMPUTES THE SQUARE
        ;ROOT OF A+B-2
        ;
```

The comment field can be extended beyond the end of the 40 character line by typing a "-" (hyphen) in position 40 of the line to be extended. The assembler will then concatenate that line with the line following. The maximum length allowed is the two

lines.

3.0 ERROR FLAGS

The Z-80 Assembler analyzes each input statement during pass 2 of the assembly process and is capable of detecting a large number of errors. The assembler uses a single character at the left of the listing line to flag a statement containing an error.

When the opcode is recognizable, the assembler generates the correct number of bytes of data and the correct binary value for the opcode. Zeros are assembled for the operand data when it is unrecognizable, so that it is possible to patch the error if the program is loaded for testing.

The error flags and their meanings are described in the following sections.

3.1 N

Error detected in label (name) field. The symbol in the label field contains more than eight characters or contains invalid characters.

3.2 O

Error detected in opcode field. The opcode was not recognized.

3.3 P

Error detected in operand field. There are a number of syntax errors which cause this flag. They are:

1. Incorrect number of operands
2. Register operands not valid
3. Operand omitted when required
4. Unbalanced quotes on ASCII string
5. Improper syntax in expression
6. Missing or unbalanced parenthesis
7. Invalid operator
8. Relative branch beyond the allowed range

3.4 U

Undefined symbol referenced in operand field.

3.5 R

Invalid forward reference, or undefined symbol in ORG, EQU, or DS instructions. All valid references to a symbol which was incorrectly forward-referenced will also be flagged with an R, but may be ignored.

3.6 D

The symbol in the location field has been defined more than once. Each occurrence of the duplicated symbol will be flagged.

3.7 A

This error will only be flagged in the QUICKRUN system when an attempt is made to load the object program into protected memory.

4.0 ASSEMBLER EXECUTION

The Z-80 Assembler is provided as a stand-alone tape, as part of the QUICKRUN package or as an executeable disk file. The operation of the disk version of the Assembler is described in the "Disk Operating System Reference Manual. This manual describes operation of the tape and QUICKRUN Assemblers.

Execution of the FUTUREDATA Assembler involves loading, and then running pass 1 followed by pass 2. Assembly of more than one program, after the assembler is loaded, can be performed easily. Also, a number of options are available which may be selected during pass 2. The following sections describe the procedures involved in running the assembler.

4.1 LOADING THE ASSEMBLER

Loading the FUTUREDATA Assembler is very simple. (Note: the QUICKRUN assembler is loaded as part of the QUICKRUN package and is always resident in the system). The assembler program tape is placed in cassette unit 1 and the "PLAY" key is pressed to prepare the unit for reading. Next, the "LOAD" button on the keyboard is pressed, followed by a "T". The cassette should then begin moving, and after the leader is passed (approximately 10 seconds), program loading will begin.

To indicate that loading is in progress, the "ENABLED" status indicator will periodically blink. If a checksum error is detected, the processor will halt, which is indicated by the "RUN" indicator going dark. If an error is encountered, rewind the program tape and repeat the above loading procedure.

Once the program has successfully loaded, a "REWIND INPUT" message will appear and the load tape may be rewound and removed. Subsequently, pressing the carriage-return will initiate execution of the assembler.

4.2 RUNNING PASS 1

After loading the assembler and once execution is initiated, the CRT will display the message "FUTUREDATA Z-80 ASSEMBLER -- VER 1.0.". The next line contains a ">" character and the assembler awaits keyboard input. The user then loads the program source tape in cassette unit 1 and prepares the unit for reading. (Note: when using the QUICKRUN system assemblies may be run on the data in the edit buffer). Typing a list of the desired pass 2 options and a carriage-return initiates execution of pass 1. When the input tape has been completely processed, the message "REWIND INPUT" is displayed. Following manual rewind of the input tape, type a carriage-return. The assembler then begins to execute pass

2. (Note: when running QUICKRUN assemblies from the edit buffer there will be no rewind messages).

4.3 RUNNING PASS 2

Normally the pass 2 assembly options are selected prior to executing pass 1. However pass 2 may be run multiple times without re-running pass 1. In this case the options are re-specified. The pass 2 options (whether specified during pass 1 or pass 2) are described below.

During pass 2 of the assembler execution, a number of options are available which are selectable by the operator. Options include listing the program on the CRT, listing the program on a hardcopy terminal, listing only lines with errors flagged, and writing an object tape.

Pass 2 options are selected when the assembler responds with the ">" character after loading the assembler or after an assembly is complete. Options are specified by typing the letter corresponding to the desired options. Option selection letters may be typed in any order. Corrections to the option list is accomplished using the cancel (CAN or CTL X) key. This deletes all options specified and provides for complete reselection of options. If an invalid option selection character is typed, a "?" appears on the display and the character is ignored.

It is not necessary to rerun pass 1 unless a new program is to be assembled. In this case a "2" should always be typed to avoid re-running pass 1. Other options are as follows:

Option Select Char	Function
L	Produce program listing on CRT. Unless specified, no listing is produced.
T	Produce a truncated listing on CRT. Line will be limited to 40 characters so that each program statement occupies only one display line. If printer option (P,B or K) is specified then DC statements will print only one line.
E	List and/or print lines with an error flag.
Z	This option will flag all lines

with non-8080 or 8085 opcodes. These lines will be flagged with a "P".

I	Input from tape. this applies only to the QUICKRUN system.
N	Inhibit object program loading into system memory. This applies to the QUICKRUN system.
P	Send a copy of the listing to the hardcopy printer (Centronics with form-feed option). "E" will cause error lines only to be printed.
P1	Print on Centronics printer without form-feed option.
K	Print on Okidata CP-110 printer.
O	Requests that an object tape be written on cassette unit 2.
B	Select hardcopy output to the serial EIA line. Normally, hardcopy data will be routed to the "Centronics" printer. The "B" option allows a wide range of printers to be used for program listing.

The character following the "B" selects the Baud rate of the serial data between 50 and 1800. If a rate is not specified then the rate specified for the previous assembly will remain in effect. The default rate, upon loading the assembler, is 110 baud. A second parameter, after B, (0 to 9) selects the number of null characters sent after a carriage return.

After "B" only:

0	Select 50 Baud for hardcopy

	printer.
1	Select 75 Baud
2	Select 110 Baud
3	Select 134.5 Baud
4	Select 150 Baud
5	Select 300 Baud
6	Select 600 Baud
7	Select 900 Baud
8	Select 1200 Baud
9	Select 1800 Baud
0-9	The second argument after the "B" option selects the number of delays after a carriage-return.

There are no restrictions on the use of the options, (except the two parameters after the "B" option) and all assembling options may be requested simultaneously. However, pass 2 may be run more than once if necessary. Once the symbol table is built during pass 1, it is not modified by execution of pass 2. Consequently, if desired options are forgotten or if it is desired to check for errors before a long printout is requested, pass 2 may be run once with one set of options and then run again with a different set of options.

So, to execute pass 2 after the options are selected, the input tape is prepared for reading on unit 1 and an object tape is prepared for writing on unit 2 (when object is requested), then execution may begin by typing a carriage-return. (Note: when using the QUICKRUN system the use of the tapes for source input or object output is optional).

At the end of assembly, the message "REWIND INPUT" is displayed. (Note: for the QUICKRUN system no rewind messages will be generated unless tape input and/or output is selected). If no errors have been detected then the message "NO ERRORS" will be displayed, otherwise the message "ERRORS FLAGGED" will appear. When the input tape is rewound, carriage-return is pressed, and if an object tape was requested, the message "REWIND OBJECT" will be displayed. Again after rewinding and typing carriage-return, the assembler prompts with a ">" symbol. At this point, pass 2 may be run again with different options, or the pass 1 of a new assembly may be initiated.

Hardcopy data will be routed to the Centronics printer via the parallel 8-bit I/O port. If the Microdisk-2/M is used then the printer is "daisy-chained" thru the disk drive. All other printers connect to the serial EIA RS-232 interface.

4.4 OPERATING SUGGESTIONS

Execution of the assembler is slowed down by requesting an object file or tape to be written and by requesting hardcopy output. Consequently, it is useful to run an assembly without either of these options on the first run before syntax errors have been removed.

On later runs, after a hardcopy has been made, if there are only a few corrections it will save considerable time to run an assembly and obtain an object tape without requesting a new hardcopy listing. Alternatively the PRNT ON and PRNT OFF pseudo instructions may be used for selective listing of the assembled output. Normally, even though there will be a few differences between the program and the listing, it is still easy to use the old listing.

While a listing to the CRT is being produced, the carriage-return key can be used to cause the assembler to suspend processing. This keeps the listing display from being rewritten while a section of the listing is examined more carefully. The assembler will respond with the message "RESTART?". If the user types "Y" carriage-return then the messages to rewind the tapes will appear. If a carriage-return is typed, then the message "CONTINUE" will appear and the assembly process will resume.

futuredata

11205 SOUTH LA CIENEGA BOULEVARD LOS ANGELES CALIFORNIA 90045 TWX 910-328-7202 TEL (213) 641-7700